Federating EdgeNet with Fed4FIRE+ and Deploying its Nodes Behind NATs

Berat Can Şenel[*,**], Maxime Mouchet[*,**],
Justin Cappos[†], Timur Friedman[*,**], Olivier Fourmaux[*], Rick McGeer[‡]
[*]LIP6-CNRS lab, Sorbonne Université, [**]LINCS lab, [†]SSL lab, NYU Tandon School of Engineering, [‡]engageLively

***Abstract*—*EdgeNet* is a globally-distributed edge cloud testbed. It aims to provide an experimentation platform to computer networks and distributed systems researchers. However, distributed testbed providers encounter difficulties with *node provisioning, access, and maintenance* when establishing an edge cloud consisting of ubiquitous nodes. *EdgeNet* extends Kubernetes to the edge and addresses these challenges. Employing such an industry-standard container orchestration system enables researchers to straightforwardly deploy experiments across many vantage points, maximizing their time for collecting and analyzing measurement data. This paper introduces three features regarding federation, home networks, and node deployment integrated into Kubernetes to handle the issues aforementioned with a free and open-source code. We also discuss remote node maintenance as future work.**

***Index Terms*—Edge Cloud, Testbed, Kubernetes, Home Network, Federation, Internet Researchers**

I. Introduction

A growing number of Internet users and their use of services such as video streaming and gaming, along with data produced by IoT devices, are placing heavy demands on cloud networks. In recent years, edge computing has been introduced to address this issue as well as to process data in real-time, which can be broadly defined as the processing of data where it is being generated [1], [2]. With edge computing comes edge clouds, which aim to provide computing services ensuring lower latency and jitter with decreased bandwidth pressure on the core network. [3].

This novel computing paradigm captures the attention of researchers in launching experiments from such environments. There is, however, a challenge for edge cloud testbed providers as they must provide and maintain nodes at the edge and access them physically and remotely, incurring additional costs. If these nodes are located behind network address translation (NAT) devices, it introduces an extra burden in terms of remote access. Taken together, these issues impair edge cloud testbeds' viability for the long term due to economic constraints.

We grouped the problems related to edge nodes into three categories: provision, access, and maintenance. Providing nodes is two-fold. First, edge computing's nature, geographically distributed environment, demands an easy procedure for node deployment. The latter is about scaling up infrastructure. There are economic and organizational obstacles to a provider establishing ubiquitous edge clouds. Nevertheless, a collaboration between edge cloud providers can lead to opening up these infrastructures to a broader community for the best use. On the other hand, physical access to nodes, primarily related to maintenance, also urges such collaboration across organizations providing nodes, whereas remote access is a technical problem. The essential issue is that nodes behind NATs do not hold a public IP address but a private IP address to communicate. Put another way, they are unreachable outside the local network by default. Therefore, a control plane of a system cannot reach out that node for taking necessary actions such as service deployment. Furthermore, edge nodes being physically inaccessible and having connectivity issues impede maintaining these nodes.

We tackle these three issues, providing, accessing, and maintaining edge nodes, through EdgeNet, a research project that consists of two pillars: the testbed[1], a globally distributed edge cloud for researchers, and the free, open-source software code on GitHub[2], which is a set of extentions to Kubernetes[3]. It already has a multitenancy framework that allows researchers to conduct experimenters in parallel, thus achieving high utilization of the cluster resources. This multitenancy framework by itself, however, is not enough to reach researchers using other testbeds. Besides, adding nodes to the cluster is already straightforward with the multi-provider feature, but a public IP address was necessary. By contrast, these features are unable to address the issues outlined earlier thoroughly.

Correspondingly, three contributions are introduced in this paper as a means of resolving the above problems:

- *Federation:* We believe a federation of edge clouds, presumably offered by different providers, is essential to establish heterogeneity and enable numerous vantage points. Such a federation can virtually address two problems discussed above: provision and maintenance. As multiple providers deliver edge nodes, a federation can ensure an infrastructure at scale. It also facilitates the maintenance of the nodes through physical operation, thanks to a shared workload between the providers.

EdgeNet got its start thanks to an NSF EAGER grant, and now benefits from a VMware Academic Program grant and a French Ministry of Armed Forces cybersecurity grant.

[1]The EdgeNet testbed https://edge-net.org/
[2]The EdgeNet software https://github.com/EdgeNet-project
[3]Kubernetes https://kubernetes.io/

We developed an aggregate manager (AM), which empowers researchers to use EdgeNet through Fed4FIRE+ [4] as they do for the other federated testbeds. That is to say, it removes the requirement of registering with the testbed to conduct measurements on our globally distributed edge cloud.

- *Home Networks:* Since the advent of edge computing, home networks have drawn heavy attention from Internet Researchers for the past few years. With home networks, NAT is an important deterrent. Thus, it is not a trivial task to manage the life-cycle of an experiment, such as deployment and version updates, that launches measurements from a node behind NAT.
  We offer a virtual private network solution that makes it possible for nodes at home networks to take part in a cluster, handling the access issue, as explained in Sec. IV. An agent running on each node configures the VPN network by actions such as assigning an IPv4/IPv6 pair of addresses. Current deployment ensures a VPN tunnel is established between two public nodes or between a public node and a NATed node. In future releases, we plan to support communication between two NATed nodes through the VPN network.
- *Node Deployment:* A fundamental necessity is a simple procedure that safely setups a node and makes it join a cluster. By providing nodes in such a forthright manner, a contributor is less likely to give up during the process. EdgeNet facilitates node contributions to the cluster via its deployment procedure. We now include the installation of the node agent mentioned above in this procedure to automate VPN establishment.

We organized the paper as follows: Sec. II presents similar testbed platforms as related works. Sec. III introduces the architecture of our Fed4FIRE+ aggregate manager, and Sec. IV describes how EdgeNet extends to home networks. In Sec. V, we describe the node deployment procedure, and Sec. VI discusses future work to maintain and recover nodes remotely. Sec. VII concludes the work.

II. Related Work

The networking and distributed systems research communities have provided various edge cloud testbeds typically spanning broad geolocations such as PlanetLab Central [5], PlanetLab Europe [6], GENI [7], G-Lab [8], V-Node [9], and SAVI [10] in the past decades. All of these testbeds required dedicated hardware and delivered custom software. These two design decisions hindered efficiency and sustainability.

First, dedicated hardware has caused an increase in maintenance and scaling costs because of a need for on-site support and initial purchase investment. Thus, contributors abandoned nodes to their fate over time. The latter is that, typically, a group of researchers writes custom software. It introduces a heavy workload on coding and preparing tutorials for those who maintain that testbed. Furthermore, it commonly obliges an experimenter to learn a new control frame for each testbed.

EdgeNet's philosophy differentiates from these testbeds in two aspects. It encourages contributors to supply virtual machines as a node instead of dedicated hardware in order to decrease the cost of providing and maintaining. To reduce programming and documentation workload, it adapts industry-standard open-source software for the needs of the testbed. Thereby, it allows to reach out to varied audiences and contribute back to the open-source community.

III. Federation

The EdgeNet testbed provides a platform for running Next-Generation Internet (NGI) experiments. It currently provides more than 50 nodes scattered around the world, with access to the Internet and private network connectivity between each nodes. EdgeNet is based on Kubernetes and experimenters deploys Docker containers. In order to provide a unified way of accessing the testbed, we implement the GENI Aggregate Manager (AM) API v3 mandated by the Fed4FIRE project. Through this API, experimenters can deploy and access containers on nodes of their choice.

Kubernetes API is natively a *declarative* API: users define the desired state of a resource (e.g. a container) and a control-loop (also called *controller*) keeps the resource in sync. On the contrary, the AM API is *imperative* by nature: users perform actions that changes the state of a resource, such as *allocate*, *provision*, *shutdown*, etc. In order to reconcile the two paradigms, we've created an AM API that manages Kubernetes objects on the behalf of the users.

A. Mapping GENI resources to Kubernetes resources

The AM specification defines three main kind of resources: users, slices and slivers. Slivers are collection of compute resources and users are given rights to create slivers in slices. In our case, we seek to offer to experimenters SSH access to Docker containers running on EdgeNet. Thus, a sliver maps to a collection of three Kubernetes objects:

- A *Deployment* object defines the specification of the container: image, node and CPU architecture. Kubernetes will ensure that a container matching these specifications will always be running.
- A *Service* object maps an available TCP port of the host node, to the SSH port of the container.
- A *ConfigMap* object holds the SSH keys of the user and is mounted on `~/.ssh/authorized_keys`.

Users can choose a specific Docker image, node and CPU architecture (aarch64 or x86_64). If none are specified, the AM API will choose a default image and Kubernetes will create the container on any available node.

B. Resource expiration

Slivers have an expiration time, which can be extended by performing the renew action. When a sliver expires, the associated resources must be deleted. Kubernetes has currently no way of specifying expiration date for objects and automatically delete them (excepted for Jobs resources). To work around this, we run a garbage collector goroutine which periodically checks for the expiration of the resources and delete them.

C. *Object naming*

Object names are derived from the first 8 bytes of the SHA512 hash of the sliver name. This allows to create objects with names that are valid in the GENI AM specification, but not in Kubernetes which allows only alphanumeric chars.

D. *Non-standard TLS certificated workaround*

As per the specification, clients are authenticated using client TLS certificated. The certificates provided by Fed4FIRE contains non-standard OIDs (Object IDentifiers) which cannot be parsed by the Go X.509 parser. Specifically, the *Authority Information Access* OID contains numbers larger than 32-bit. This makes it impossible to authenticate clients using Go code and prevents the use of reverse proxies written in Go such as the popular Caddy [11] and Traefik [12] proxies. Fed4FIRE administrators have been contacted about this issues and stated that they have no intent to change the OID format. To work around this issue, we place an NGINX [13] reverse proxy in front of our AM API server. This proxy performs client TLS authentication and forwards the request to the AM API server by including the certificate in a custom `X-Fed4Fire-Certificate` HTTP header. This information is then passed to external tools by the AM API, such as `xmlsec1` to validate credentials and authorize users.

E. *Deployment*

Our AM API is publicly deployed[4] and its source code is available on GitHub.[5] It can easily be deployed on any Kubernetes cluster to federate it with Fed4FIRE.

IV. Home Networks

Kubernetes is designed for centralized data centers, and thereby, the system assumes that cluster nodes share a local network. Put another way; it does not provide an off-the-shelf solution for nodes on different networks without public IP addresses. It introduces two problems in terms of communication:

- A node behind NAT can access the control plane nodes, but they cannot reach that node out.
- Containers on a node behind NAT are unreachable from other cluster nodes.

Kubernetes has an extensible architecture that allows developing and using plugins. A Container Network Interface (CNI) plugin typically establishes networking between pods. EdgeNet employs VMware's Antrea CNI[6] for this purpose. Antrea uses the OVS bridge on every node. Furthermore, it forms a veth pair for each pod, a gateway (gw) to the node subnet, and a tunnel (tun) for inter-node communications [14].

The OVS bridge forwards packets using veth pairs on the node regarding local pod traffic. If traffic is toward an external destination, packets to be routed are forwarded through the gateway port. Antrea benefits from Source Network Address Translation (SNAT) so that the pod IP address is preserved. In terms of inter-node communication, tunnels encapsulate and decapsulate packets. Fig. 1 depicts the traffic flow where every node has a public IP address.

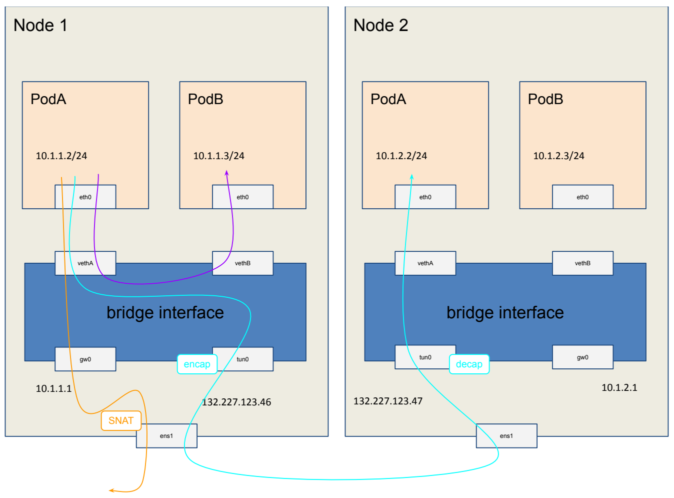

Fig. 1: Represents pod traffic where every node has a public IP address. Orange is for a pod to external, Cyan is for inter-node, and Purple is for intra-node traffic.

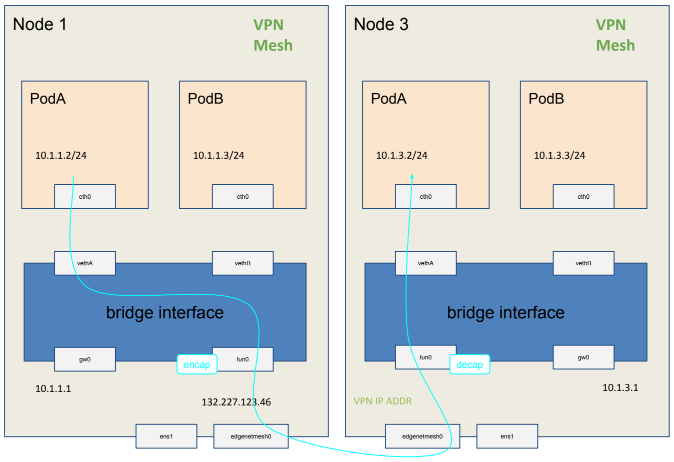

Fig. 2: Illustrates pod traffic where a node has a private IP address. Cyan shows that traffic is routed through edgenetmesh both on source and destination.

However, if a node is behind NAT on another network, it blocks inter-node communication. To overcome this problem, we setup a VPN tunnel between every pairs of nodes in the cluster. We settled on using the WireGuard VPN [15] for multiple reasons:

- its performance: it offers throughput 5 times higher than OpenVPN and 1.1 times higher than IPsec on the same configuration [16];
- its simplicity: it only requires generating of public/private pair of keys for each client and does not requires a PKI infrastructure as OpenVPN certificate-based authentication does;

[4]The AM API https://fed4fire.edge-net.io.
[5]The AM software https://github.com/EdgeNet-project/fed4fire.
[6]VMware Antrea https://antrea.io/

- its integration in the Linux kernel: it is natively integrated in the kernel since Linux 5.6, requiring no additional kernel modules.

Our solution provides IPv4 and IPv6 peer-to-peer communication for every nodes of the cluster (with the exception of NAT-to-NAT communications, see Sec. IV-B) over the public IPv4 Internet, as seen in Fig. 2.

A. *Bootstrapping VPN peers*

A node must have established VPN connectivity with the rest of the cluster before Kubernetes starts. To achieve this, an agent present on each node performs the following actions on boot:

1) It checks if a public/private key pair has ever been generated. If none, it generates one and save it for subsequent boots.
2) It checks if an IPv4/IPv6 pair of address has ever been generated. If none, it queries the cluster to get the list of used IP address in the VPN network, and choose a random pair of addresses amongst the one available. Randomization allows to reduce the risk of two new nodes choosing the same IP address if booted at the same time.
3) It publishes its public key, its private IP address pair and its public IP address to the cluster, by creating a *VPNPeer* Kubernetes resource.
4) It queries the list of *VPNPeer* resources and configure the tunnel interface to add each peer.

B. *NAT-to-NAT communications*

In our current deployment, a VPN tunnel can be established between two public nodes, or between one public node and one NATed node. Establishing a connection between two NATed nodes requires the use of an external server to exchange port numbers and perform UDP hole punching. We will implement such a technique in future iterations.

V. Node Deployment

Anyone can contribute a node to the EdgeNet public cluster. Deploying a node involves setting-up Kubernetes, the container runtime, the network and joining the cluster. In order to make this process as easy and as transparent as possible, we developed a set of Ansible playbooks[7] that automatically performs all these steps. Our current deployment procedure is as follows and will spawn a node in under 5 minutes:

- The user runs the `bootstrap.sh` on the target node. This script will install Ansible, fetch the deployment playbook and run it.
- The playbook will install Kubernetes, the container runtime, and a dedicated *node agent* written in Go.
- The node agent will start and configure the VPN as described in Sec. IV-A, and it will join the cluster.

This architecture is very flexible:

[7]Node provisioning https://github.com/EdgeNet-project/node.

- The bootstrap script works on any recent Debian or RedHat-based Linux distribution, on aarch64 or x86_64 architectures, and it doesn't requires any preinstalled software.
- The bootstrap script URL can be passed to cloud-init [17] to automatically setup the instances on first boot.
- The Ansible playbooks can be used in a standalone fashion for bulk deployment or node maintenance.
- The Ansible playbooks can be used together with Packer [18] to create VM images with our software pre-installed.

In our current implementation, no input is required from the user and anyone can contribute a node without an EdgeNet account.

VI. Node Robustness

In comparison to other testbeds such as PlanetLab, EdgeNet nodes are not expected to be maintained by system administrators. A node can be deployed in a user home with limited debugging time and knowledge. As such we must ensure that the nodes are able to self-heal in case of problems. We currently identified two main issues: unresponsive nodes and filesystem corruption. We describe below two tentative solutions that we will try to implement in the next iteration of EdgeNet.

A. *Unresponsive nodes*

A node can become unresponsive if some application consumes all of its resources, or if an excessive amount of network trafic saturates the network interface and the CPU. We are investigating the use of the hardware watchdog present on Raspberry Pi and Odroid single-board computers to automatically reboot unresponsive nodes. The kernel periodically send heartbeats to the watchdog. If the watchdog stop receiving heartbeats it power cycle the node. This procedure can fix unresponsive nodes without any user intervention.

B. *Filesystem corruption*

Single-board computers often use flash-based memory such as SD cards or eMMCs. These memory are prone to failure as they are usually not designed for permanent random writes over long period of times. When these memory fails the filesystem is corrupted and the systems stops working properly. It is also possible that a user improperly changes the configuration of a node.

To handle this issue we are exploring the possibility of booting nodes over the Internet. The petitboot [19] bootloader can boot a kernel and a live disk image over the Internet. The disk image and the kernel would live in RAM and the node's flash storage would only be used to store containers data. If the flash memory fails, the node would still be accessible and the user would only have to replace the SD card.

This method has also the benefit of making system updates very easy: deploy a new disk image on the server and reboot the remote nodes. If the update fails, rollback the disk image on the server. The main concern with this approach is security and how to authenticate the boot server as well as the disk images.

VII. Conclusion

We have introduced challenges related to nodes at the edge that distributed testbed providers face: provision, access, and maintenance. Three contributions, federation, home networks, and node deployment, addressed these issues. Our aggregate manager (AM), which federates EdgeNet into Fed4FIRE+, shows that our testbed is capable of running in concert with other testbeds. In addition to this, a node agent configures a virtual private network, thus enabling nodes behind NATs to participate in a cluster with the help of the native VPN peer controller in Kubernetes. The last one is a node deployment procedure that facilitates making a node join a cluster in under ten minutes, including installing the node agent mentioned above. In conclusion, these three features allow the testbed to reach out to a wider community and scale up the cluster, including edge nodes typically blocked by NAT. This paper also discusses how to tackle remotely maintaining edge nodes as future work.

Acknowledgment

EdgeNet got its start thanks to an NSF Eager grant, and now benefits from a VMware Academic Program grant and a French Ministry of Armed Forces cybersecurity grant.

References

[1] W. Shi and S. Dustdar, “The promise of edge computing,” *Computer*, vol. 49, no. 5, pp. 78–81, 2016. [Online]. Available: https://doi.org/10.1109/MC.2016.145

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016. [Online]. Available: https://doi.org/10.1109/JIOT.2016.2579198

[3] M. Satyanarayanan, “The emergence of edge computing,” *Computer*, vol. 50, no. 1, pp. 30–39, 2017. [Online]. Available: https://doi.org/10.1109/MC.2017.9

[4] Fed4FIRE+, “Fed4fire+ testbeds portal,” 2022. [Online]. Available: https://www.fed4fire.eu/

[5] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir, “Experiences building planetlab,” in *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, ser. OSDI ’06. USA: USENIX Association, 2006, p. 351–366. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/1298455.1298489

[6] S. Fdida, T. Friedman, and T. Parmentelat, “OneLab: An open federated facility for experimentally driven future internet research,” in *New Network Architectures: The Path to the Future Internet*, ser. Studies in Computational Intelligence, T. Tronco, Ed. Springer-Verlag, 2010, vol. 297, pp. 141–152. [Online]. Available: https://doi.org/10.1007/978-3-642-13247-6

[7] R. McGeer, M. Berman, C. Elliott, and R. Ricci, Eds., *The GENI Book*. Springer International Publishing, 2016. [Online]. Available: https://doi.org/10.1007/978-3-319-33769-2

[8] P. Mueller and S. Fischer, “Europe’s mission in next-generation networking with special emphasis on the German-Lab project,” in *The GENI Book*. New York: Springer-Verlag, 2016, ch. 21. [Online]. Available: https://doi.org/10.1007/978-3-319-33769-2_21

[9] A. Nakao and K. Yamada, *Research and Development on Network Virtualization Technologies in Japan: VNode and FLARE Projects*, 09 2016, pp. 563–588. [Online]. Available: https://doi.org/10.1007/978-3-319-33769-2_23

[10] A. Leon-Garcia and H. Bannazadeh, “SAVI testbed for applications on software-defined infrastructure,” in *The GENI Book*. New York: Springer-Verlag, 2016, ch. 22. [Online]. Available: https://doi.org/10.1007/978-3-319-33769-2_22

[11] S. Holdings. (2022, 3) Caddy 2 - The Ultimate Server with Automatic HTTPS. [Online]. Available: https://caddyserver.com

[12] T. Labs. (2022, 3) Traefik. [Online]. Available: https://doc.traefik.io/traefik

[13] F. Networks. (2022, 2) Advanced Load Balancer, Web Server, & Reverse Proxy - NGINX. [Online]. Available: https://www.nginx.com

[14] T. A. Contributors, “Antrea architecture,” 2022. [Online]. Available: https://antrea.io/docs/v1.5.0/docs/design/architecture/

[15] J. A. Donenfeld, “WireGuard: Next Generation Kernel Network Tunnel,” in *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/wireguard-next-generation-kernel-network-tunnel/

[16] ——. (2022, 3) Performance - WireGuard. [Online]. Available: https://www.wireguard.com/performance

[17] C. Ltd. (2022, 1) CloudInit - The standard for customising cloud instances. [Online]. Available: https://cloud-init.io

[18] HashiCorp. (2022, 3) Packer by HashiCorp. [Online]. Available: https://www.packer.io

[19] open power. (2022, 3) petitboot. [Online]. Available: https://github.com/open-power/petitboot